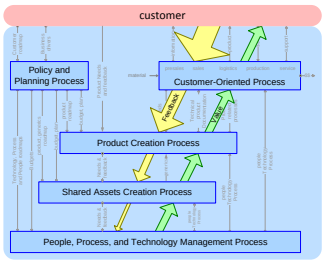
Product Families and Generic Aspects

-



Gerrit Muller
University of Southeast Norway-NISE
Hasbergsvei 36 P.O. Box 235, NO-3603 Kongsberg Norway
gaudisite@gmail.com

This paper has been integrated in the book "Systems Architecting: A Business Perspective", http://www.gaudisite.nl/SABP.html, published by CRC Press in 2011.

Abstract

Most products fit in a larger family of products. The members of such a product family share a lot of functionality and features. It is attractive to share implementations, designs et cetera between those members to increase the efficiency of the entire company.
In practice many difficulties pop up when product developments become coupled, due to the partial developments which are shared. This article discusses the advantages and disadvantages of a family approach based on shared developments and provides some methods to increase the chance on success.

Distribution
This article or presentation is written as part of the Gaudí project. The Gaudí project philosophy is to improve by obtaining frequent feedback. Frequent feedback is pursued by an open creation process. This document is published as intermediate or nearly mature version to get feedback. Further distribution is allowed as long as the document remains complete and unchanged.

All Gaudí documents are available at:
http://www.gaudisite.nl/

version: 2.3 status: concept June 5, 2018

1 Introduction

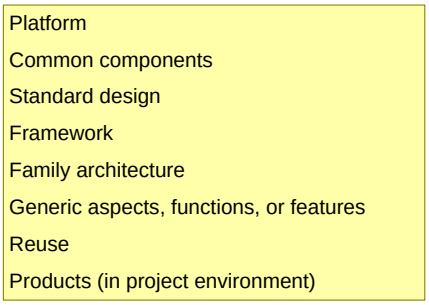


Figure 1: Different names for development strategies that strive to harvest synergy

Harvesting synergy between products or projects is being done under many different names, such as shown in Figure 1. We use *generic developments* or *harvesting synergy* as label for this phenomena. The reader may substitute the name that is used in their organization.

Many trends (increased variability, increased number of features, increased interoperability and connectivity, decreased time to market, globalization of development, globalization of markets) in the world force organizations into these strategies where synergy is harvested. Harvesting synergy is, however, also a complicating factor both organizational and technical. We strive to give insight in both needs and complications of harvesting synergy, in the hope that awareness of the complications will help to establish an effective synergy harvesting strategy.

2 Why generic developments?

Many people advocate generic developments, claiming a wide range of advantages, such as listed in Figure 2.

Effective implementation of generic development has proven to be quite difficult. Many attempts to achieve these claims by generic developments have resulted in the opposite of these claims and goals, such as increased time to market, quality and reliability problems et cetera. We need a better rationale to do generic developments, in order to design an effective Shared Assets Creation Process.

Figure 3 shows drivers for Generic Developments and the derived requirements for the Shared Assets Creation Process. The first driver (*Customer value* is extrovert: does the product have value for the customer and is the customer willing to buy the product? The second driver *Internal Benefits* is introvert, it is the normal economic constraint for a company.

Today high tech companies are know how and skill constrained, in a market

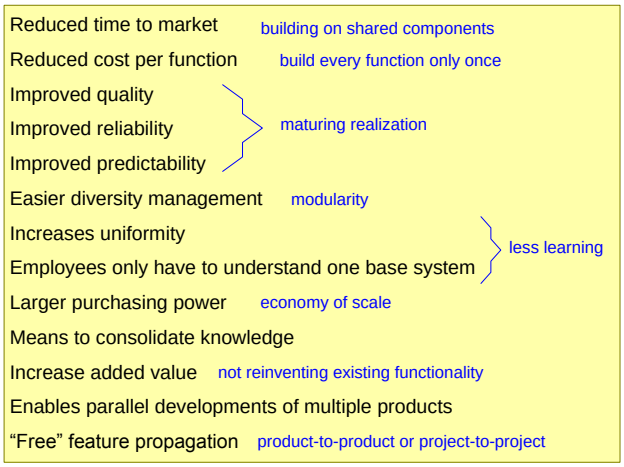


Figure 2: Advantages which are often claimed for generic developments

that is extremely fast changing and rather turbulent. Cost considerations are an economic constraint that has to be balanced with the capability to create valuable and sellable products.

The derivation of the requirements for the product development shows that these requirements are not a goal in itself, but are a means to facilitate an higher level goal. For instance, a shared architecture framework is required to enable features developed for one product to be used in other products too. This propagation of features makes sense if it creates value for a customer. So the verification of the shared architecture framework requirement has to involve the propagation a new feature from one product to the next, using limited effort and lead time.

We emphasize the derivation from drivers to requirements because many generic developments fulfil the requirements, such as *availability accumulated feature set*. *designed for configurability*, *shared architectural framework*, and *maturity or implementation*, without bringing the assumed customer or sales value. For example, many generic developments result in large monolithic solutions, without flexibility and long development times. Developers of such framework have been providing replies as: ”You can not have this easy shortcut, because our architectural framework does not support it, changing the framework will cost us 100 man-years in 3 years elapsed time”.

3 Granularity Of Generic Developments

Granularity is one of the key design choices for systems architects: what is an appropriate decomposition level for modularity? Granularity decisions have to be made at all levels for different purposes. For example, in the application granularity

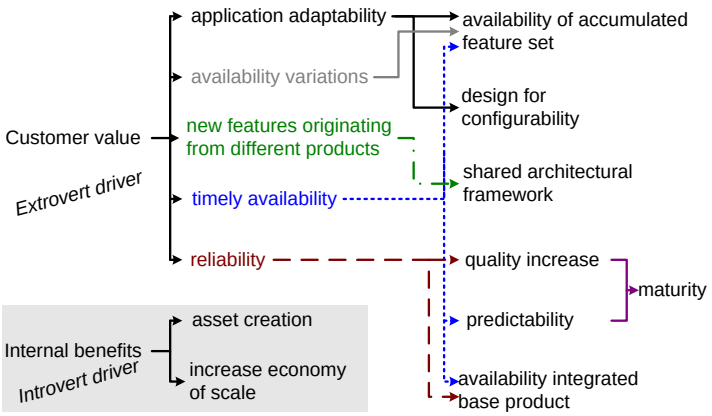


Figure 3: Drivers of Generic Developments

of functions and roles, at specification level granularity of options and features, in conceptual design granularity of functions and concepts, and in implementation granularity of many operations.

Figure 4 shows the granularity of generic developments in 2 dimensions. The vertical dimension is the preparation level: What is the intended scope of the generic developments, how far is the deployment prepared? The horizontal dimension is the integration level: How far are the generic developments integrated when the *product developers* deploy the generic development?

Both axis range from (atomic) component until (configurable) system. Developments on the diagonal axis, which have a scope where the preparation level is equal to the integration level, are straightforward developments in which the integration takes place as far as autonomously possible. Some generic developments concentrate on the generation of building blocks, leaving ("delegating") the integration to the product developer. For rather critical generic developments the the integration of the shared asset goes beyond its own deliverable to ensure the correct performance of the asset in its future context(s).

In these figures a number of medical generic developments are shown, as an example for the categorization.

An extreme example of "delegated" integration is Common Viewing (CV). The organization made an attempt to harvest synergy at the end of the eighties. The vision was to create a large "toolbox" with building blocks that could be used in a wide variety of medical products ranging from Magnetic Resonance Imaging (MRI) scanners to X-ray systems. A powerful set of (mostly SW) components was created, using Object Oriented technology and supporting a high degree of configurability

The CV toolbox proved difficult to sell to product developers, amongst others

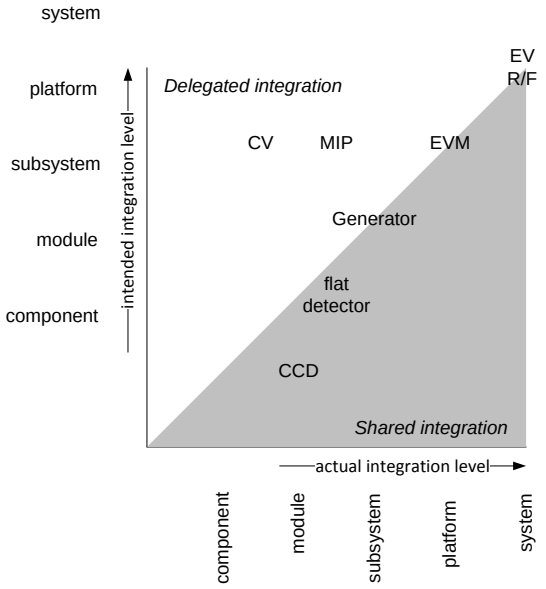


Figure 4: Granularity of generic developments shown in 2 dimensions.

due to the low integration level. The perception of the product developers was that they still had to do the majority of difficult work: the integration. The vision of a marketing manager changed the direction of CV into creating a completely integrated product: EasyVision Radiography Fluoroscopy (EV RF). This medical workstation for the URF (Universal Radiography Fluoroscopy) market was highly successful, serving as an intelligent print server. The communication and print function were highly configurable to make the product adaptable to its environment.

The EasyVision RF was used as a basis for a whole series of medical workstations and servers. The shared functionality is developed as generic development at platform level. This platform is nowadays called EasyVision Modules (EVM). Despite its name it has still a significant integration level, with its upside (product developers are not bothered with the lower level integration) and its downside (predefined functionality and behavior).

The old CV vision is revived and a second generation of EVM is being created, covering the EVM platform functionality with finer granularity: a module level of integration. The whole evolution as described here from CV as toolbox to more fine grained EVM modules took about 15 years. During all these years the balance between genericity (degree of sharing) and customer value has been changing without ever achieving the combination of a high degree of sharing and a high customer value at the same time.

4 Modified Process Decomposition

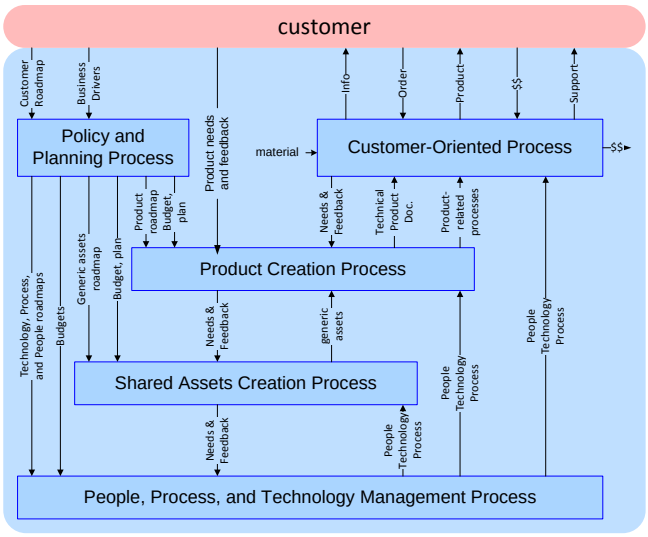


Figure 5: Modified process decomposition

In **??** we discussed a simplified process description of companies. This decomposition assumes that product creation processes for multiple products are more or less independent. When generic developments are factored out for strategic reasons then an additional process is added: the Shared Assets Creation Process. Figure 5 shows the (still simplified) modified process decomposition

Figure 6 shows these processes from the financial point of view. From financial point of view the purpose of this additional process is the generation of strategic assets. These assets are used by the Product Creation Process to ensure the cash flow for the near future by staying competitive.

The consequence of this additional process is an lengthening of the value chain and consequently a longer feedback chain as well. This is shown in figure 7. The increased length of the feedback chain is a significant threat for generic developments. The distance between designers and developers of shared assets and the stakeholders in the outside world is large. These developers easily lose focus on customer value and may focus on the technology instead. Successful sharing requires a strong relation between customer value and technology.

5 Modified Operational Organization of Product Creation

The operational organization of the Product Creation Process is described in **??**. This organization is a straightforward hierarchy, where the limited amount of relations

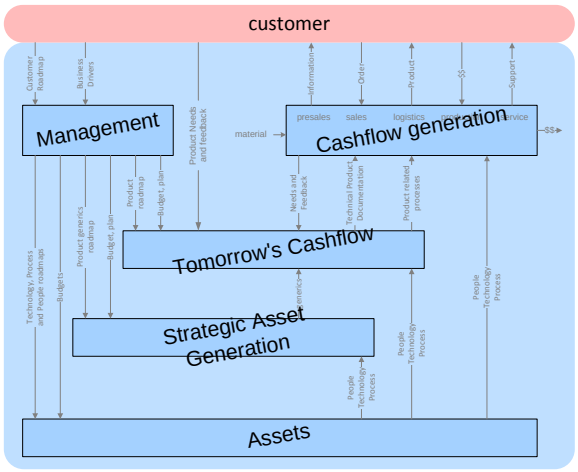


Figure 6: Financial viewpoint of processes

(conflicts) between products or subsystems are managed at the closest hierarchical management level.

Introduction of generic developments complicates the operational structure significantly[1]. Figure 8 shows the operational organization of the Product Creation Process, with the necessary additions to support generic developments.

The conventional Product Creation Process is based on a relative straightforward hierarchy, where the control flow and delivery flow are opposite, where both flows follow the hierarchy. The introduction of generic developments breaks this simple structure: a generic development team delivers to multiple product developments, where the control is taking place from an encompassing operational level, to enable operational balancing of products and generic developments. In other words the principal of the project leader is not the customer anymore, but an intermediate manager.

Every operational entity needs the 3 complementing processes in the product creation process: operational management, design control and commercial. For each of these processes a role is required of someone responsible for that process: the operational manager, the architect and the commercial manager. Together these 3 people form the core team of the operation. Introduction of generic developments also requires the introduction of these roles for the shared assets, such as platform or components.

For the architect role this means that a platform architect is needed, who is closely working together with the platform project leader and the platform manager.

[1]The complication can be avoided by working sequentially. However in today's dynamic market sequential work results in unacceptable lead times. Concurrent engineering is a fact of life. Organizations are looking for opportunities to reduce the lead time more.

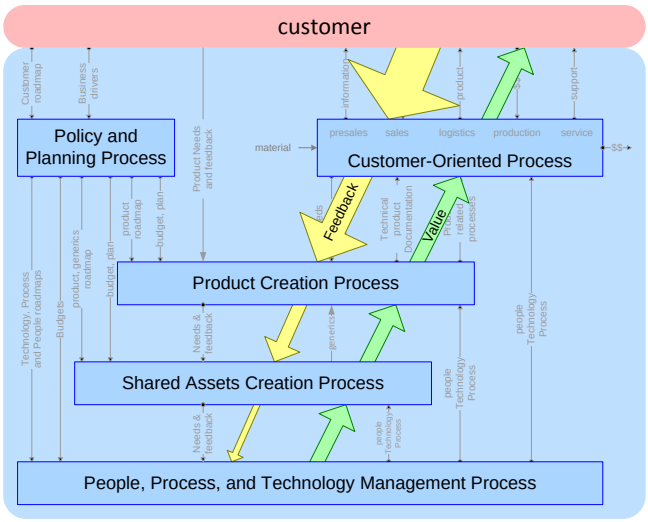


Figure 7: Feedback and Value flow

At the other hand the platform architect needs many architectural contacts with the product family architect, acting as the architectural principal, with the product architect, acting as customers, and with the component architects, acting as suppliers.

The separation of the roles of the platform architect and the product family architect is not obvious. For example in [1] 3 operational entities with related processes and roles are identified. Application Family Engineering (AFE), Component System Engineering (CSE), and Application System Engineering (ASE) map respectively on Product Family, Component, and Product as shown in Figure 8. We will either have a gap or a double role, when mapping 4 operational entities on 3 processes. In practice the result is that one of the roles is missing, or played implicit. For instance quite often the application family engineer starts to play platform architect, forgetting the original task *application family* engineering. We have observed that architects either tend to play the platform architect role or the product family role. Architects combining both roles naturally are scarce.

6 Models for Generic Developments

Many different models for the development of shared assets are in use. An important differentiating characteristic is the driving force, often directly related to the de facto organization structure. The main flavors of driving forces are shown in figure 9.

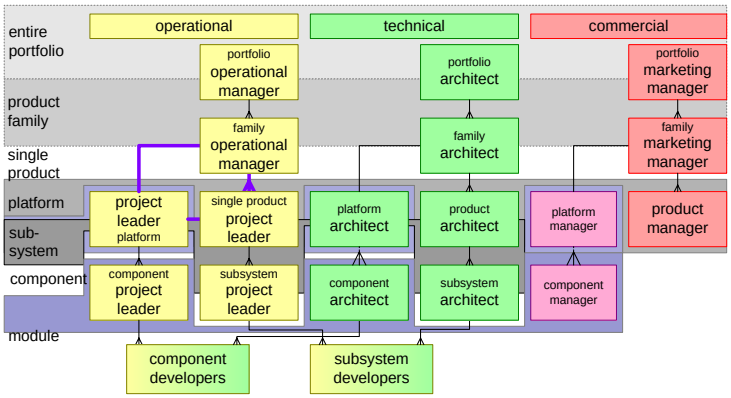


Figure 8: Operational Organization of the Product Creation Process, modified to enable generic developments

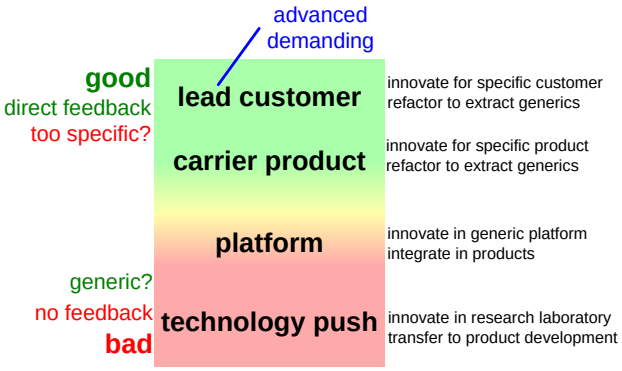


Figure 9: Models for SW reuse

6.1 Lead Customer

The lead customer as driving force guarantees a direct feedback path from an actual customer. Due to the importance of feedback this is a very significant advantage. The main disadvantages of this approach are that the outcome of such a development often needs a lot of work to make it reusable as a generic product. The focus is on the key functions and performance parameters of the lead customer, while all other functions and performance parameters are secondary in the beginning. Also the requirements of this lead customer can be rather customer specific, with a low value for other customers.

6.2 Carrier Product

The combination of a generic development with one of the product developments also shortens the feedback cycle, although the feedback is not as direct as with the lead customer. Combination with a normal product development will result in a better coverage of performance parameters and functionality. Disadvantage can be that the operational team takes full ownership for the product (which is good!), while giving the generic development second priority, which from family point of view is unwanted.

In larger product families the different charters of the product teams create a political tension. Especially in immature or power oriented cultures this can lead to horrible counterproductive political games.

Lead customer driven product development, where the product is at the same time the carrier for the platform combines the benefits of the lead customer and the carrier product approach. In our experience this is the most effective approach of generic developments. A prerequisite for success is an open and result driven culture to preempt any political games.

6.3 Platform

Generic developments are often decoupled from the product developments in maturing product families, by creating an autonomous Shared Asset Creation Process. In products where integration plays a major role (nearly all products) the shared assets are pre-integrated into a platform or base product. Such platform or base product follows its own release process before it can be used by product developments.

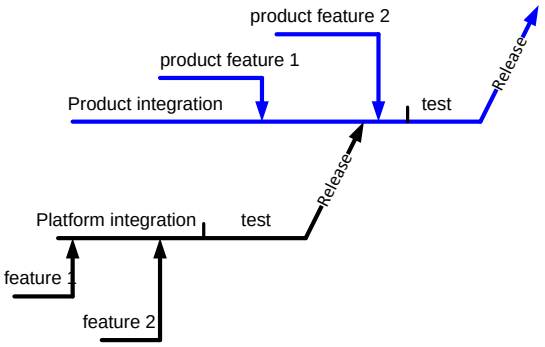


Figure 10: The introduction of a new feature as part of a platform causes an additional latency in the introduction to the market.

The benefit of this approach is separation of concerns and decoupling of products and platforms in smaller manageable units. These benefits are also the main weakness

of such a model: as a consequence the feedback loop is stretched to a dangerous duration. At the same time the duration from feature/technology to market increases, see figure 10.

6.4 Alternative Generic Development Scenarios

A number alternative re-use strategies have been applied with more or less success:

Spin-out as an independent company is especially tried for key and base technologies. However, many spin-out companies have been re-absorbed by their parent companies. Examples are multimedia processors from TriMedia (parent Philips Semiconductors, later NXP) and cell phone operating system Symbian (parent Nokia)

Reuse after use works quite good in practice, especially for good clean designs.

Opportunistic copy where implementations are taken that are available. The results are quite mixed. Short term benefits are quick results and hence short feedback cycles. Longer term a problem can be that an architectural mess has been growing that turns into a legacy.

Open source where key and base technologies are shared and developed much more publicly.

Inner-source , where a company stimulates sharing takes place within a company modeled after an open source approach.

volutionary refactoring where the architecture and its components are actively re-factored to keep them fit for the future and for potentially increased scope of application

7 Common Pitfalls

We learn from out mistakes. Unfortunately, many mistakes have been made in the area of generic developments. We compiled the list of pitfalls shown in Figure 11 from mistakes in generic developments in the past. Some of the attempts to harvest synergy were partially successful, but issues from the list limited the degree of success.

Most of the problems have a root cause in people, process, or organizational issues. The list with technical problems is relatively small:

Too generic platform or components that can do everything, but nothing really good: “the Swiss army knife”

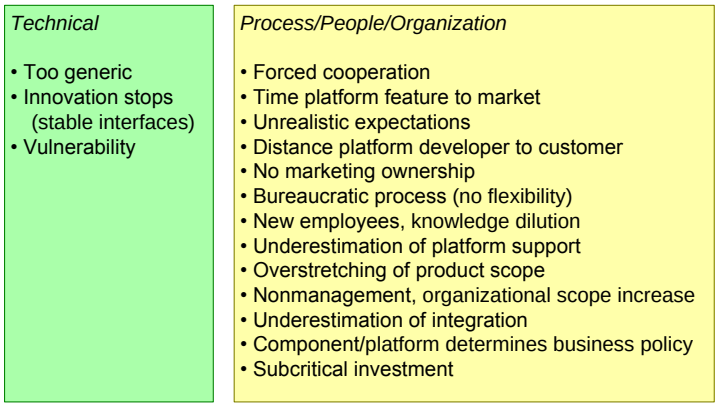


Figure 11: Sources of failure in generic developments

Innovation stops , because existing interfaces are declared to be stable. Existing structure and interfaces can block innovation.

Vulnerability , because all products use one and the same core. If the shared core has a problem anywhere then all products are hit simultaneously. Diversity is a characteristic that enhances resilience. In nature, species often survive disasters, such as diseases, due to the diversity in the population.

Forced cooperation by upper management, de-motivating employees, and creating social and political tensions in the organization.

Time platform feature to market because of stacked release procedures.

Unrealistic expectations by upper management, often as a consequence of the claims from architects and engineers o the benefits of harvesting synergy. When less is delivered than promised, then a negative spiral sets in of cost reduction and hence even more decreasing outcome.

Distance platform developer to customer , see Figure 7.

No marketing ownership , but engineering push only. Marketing support is crucial, since marketing is one of the key players when making decisions about investments. Lack of marketing ownership results in a continuous fight for funding, with starvation in the end.

Bureaucratic process , and loss of flexibility. The increased scope of the operation (common components or platform plus derived products) often requires a more formal organization than the individual products used to have. The formalization easily turns into bureaucratism, slowing down the entire organization.

Knowledge dilution caused by the hiring of new employees. Often an increase in resources is needed early during the creation of shared assets. If these new resources are inexperienced, then the knowledge is diluted, resulting in less quality of the created assets.

Underestimation of shared asset support required when the shared assets are used by products. Product designers need support when specifying and designing new products based on these assets, and they need support for trouble shooting during integration and introduction in the field. When components are used in new circumstances (e.g. new products), then always unexpected problems pop-up.

Overstretching of product scope beyond the natural level of synergy. Harvesting synergy is a balancing act, between maximum value creation for specific customers and minimizing diversity in the realization. When the minimization of diversity dominates over value creation, then customers are not served well, resulting in a loss of business. Organizations easily lose their customer focus, when creating a synergy drive.

Non-management of organizational scope increase that is inherent when multiple products share assets. The scope increase requires organization, process and staffing adaptations.

Underestimation of integration of shared assets in other products. Systems integration is often ill understood and hence underestimated, see **??**. When existing products have to migrate to the use of shared assets, then this requires that these products adapt their architecture too.

Component/platform determines business policy which is effectively an inversion of the need driven approach. This inversion relates to the distance between shared asset development and customers. What happens is that what *can* be done dominates over what *needs* to be done. The shared asset developers get de facto power, since all products depend on their delivery.

Subcritical investment , caused by a cost reduction focus. Shared asset development primarily should bring market and customer value, while keeping the cost limited by harvesting synergy. As soon as cost reduction dominates over value creation, then all products and shared assets can get too little investment, causing delays and quality problems.

8 Acknowledgments

During the first CTT course system architecture, from november 22 until november 26 1999, a lively discussion about generic developments took place, which created

a lot of input for this article. I am grateful to the following people, who attended this course: Dieter Hammer, Wil Hoogenstraaten, Juergen Mueller, Hans Gieles, Huib Eggenhuisen, Maurice Penners, Pierre America, Peter Jaspers, Joost Versteijlen, Peter Beelen, Jarl Blijd, Marcel Dijkema, Werner Roelandt, Paul Janson, Ashish Parasrampuria, Mahesh Bandakka, Jodie Ledeboer

I thank Pierre America for working on consistency in spelling and the use of capitols. Ad van den Langenberg pointed out a number of spelling errors.

References

[1] Ivar Jacobson, Martin Griss, and Patrik Jonsson. *Software Reuse; Architecture, Process and Organization for Business Success*. ACM Press, New York, 1997.

[2] Gerrit Muller. The system architecture homepage. `http://www.gaudisite.nl/index.html`, 1999.

History

Version: 2.3, date: July 29, 2010 changed by: Gerrit Muller

- textual updates
- changed status to concept

Version: 2.2, date: June 8, 2010 changed by: Gerrit Muller

- replaced lists by figures
- removed Griss diagram
- removed alternative re-use scenarios

Version: 2.1, date: December 13, 2007 changed by: Gerrit Muller

- added list with experiences to presentation

Version: 2.0, date: March 2, 2005 changed by: Gerrit Muller

- update of figures

Version: 1.5, date: March 2, 2005 changed by: Gerrit Muller

- improved text about operational organization

Version: 1.4, date: March 6, 2003 changed by: Gerrit Muller

- updated figure modified operational organization
- replaced table reuse models with figure

Version: 1.3, date: August 5, 2002 changed by: Gerrit Muller

- small editorial changes only

Version: 1.1, date: September 12, 2001 changed by: Gerrit Muller, Pierre America

- small editorial changes only

Version: 1, date: June 20, 2001 changed by: Gerrit Muller

- updated layout
- added list of platform pitfalls
- added list of alternative re-use scenario's
- moved most of introduction into abstract

Version: 0, date: January 13 2000 changed by: Gerrit Muller

- Created, no changelog yet